Utilizing Optimization Potential during Multicode Identification

Ben Stephenson and Wade Holst
University of Western Ontario, London, Ontario, Canada
ben@csd.uwo.ca   wade@csd.uwo.ca

Abstract

Multicodes have been demonstrated to provide performance gains of up to 25 percent [1]. Present multicode identification techniques rely on frequency of occurrence and sequence length alone. This research extends previous work by presenting a multicode identification algorithm based on frequency of occurrence and optimization potential of the sequence.

1 Introduction to Multicodes

Multicodes improve Java interpreter performance:

- Profiling is used to identify bytecode sequences that are executed frequently
- A new multicode is introduced that provides identical functionality
- All occurrences of the sequence are replaced with the new multicode
- The multicode offers superior performance compared to the sequence
  - Transfers of control from one bytecode to the next are overhead – using a multicode decreases the number of transfers of control
  - Additional optimization opportunities become available when the separating transfers of control are removed

Consider the bytecodes executed to evaluate the expression $y = mx + b$:

... → fload_1 → fload_2 → fmul → fload_3 → fadd → fstore_0 → ...

A new multicode is introduced with an opcode that is not currently defined by the Java Virtual Machine Specification. The number of transfers required for this computation has been reduced from 6 to 1.

... → fload_1/fload_2/fmul/fload_3/fadd/fstore_0 → ...

The code for this new multicode is shown below:

```
... →  fload_1:            fload_2:            fmul:                              fload_3:            fadd:                              fstore_0:
         top++;              top++;              st[top-1] = st[top] * st[top-1];   top++;              st[top-1]=st[top] + st[top-1];     vars[0] = st[top]; → ...
       st[top] = vars[1];  st[top] = vars[2];    top–;                            st[top] = vars[3];    top–;                              top–;
```

Optimizations can be performed across the codelet boundaries since there are no longer intervening transfers of control.

1: Collapse All Stack Adjustments	2: Perform Copy Propagation	3: Remove Unnecessary Stack Writes
st[top+1] = vars[1];	st[top+1] = vars[1];	
st[top+2] = vars[2];	st[top+2] = vars[2];	
st[top+1] = st[top+2] * st[top+1];	st[top+1] = vars[2] * vars[1];	st[top+1] = vars[2] * vars[1];
st[top+2] = vars[3];	st[top+2] = vars[3];	
st[top+1] = st[top+2] + st[top+1];	st[top+1] = vars[3] + st[top+1];	st[top+1] = vars[3] + st[top+1];
vars[0] = st[top+1];	vars[0] = st[top+1];	vars[0] = st[top+1];
// no stack adjustment needed		

It is possible to perform copy propagation on more complex expressions making additional stack writes unnecessary. The resulting codelet for fload_1/fload_2/fmul/fload_3/fadd/fstore_0 is:

vars[0] = vars[3] + vars[2] * vars[1];

Other multicodes do not offer as much optimization potential. As a result, a multicode selection technique must be developed that considers more than just the number of transfers of control that will be removed.

2 Multicode Identification

Previous multicode identification techniques computed the score of each multicode as the total number of transfers that would be eliminated (frequency × (length - 1)). This fails to consider:

- The optimization potential of the sequence
- The impact introducing the new multicode has on
  - changes in cache performance / code locality
  - optimization within the multicode codelet
  - decoding the opcode and branching to reach the correct codelet

We attempted to use micro-benchmarking to estimate these costs. However problems were still encountered.

- Difficult to micro-benchmark multicodes that contain return bytecodes
- Fails to consider the cumulative impact of multiple multicode substitutions
- Did not offer superior performance to previous technique

To overcome these difficulties, we elected to use a technique that involves timing the benchmark itself to determine the effectiveness of candidate sequences.

- Five best candidates are selected based on the number of transfers that will be removed
  - Transfer removal represents a significant proportion of the cost saving achieved by using multicodes
  - The number of transfers that will be removed is easy to calculate
  - Taking these factors together, transfers removed is a good choice for the identification of candidates to be evaluated further
- Each of the candidates identified is timed. By performing timing the overall impact is measured including the impact of caching and multicode optimization.
- The candidate with the lowest runtime is selected.
- The process is repeated until the desired number of multicodes has been identified.
  - Each new timing includes all of the multicodes identified before it. This ensures that any interaction between the multicodes is also measured.
- This technique has been used to speed application performance by as much as 30% for _222_mpegaudio.
- Performance results achieved when timing is used during multicode identification are consistently better than performance results achieved when multicodes are identified by transfer reduction alone

The top multicodes identified by the previous transfer reduction technique and the new timing technique are shown in the table below.

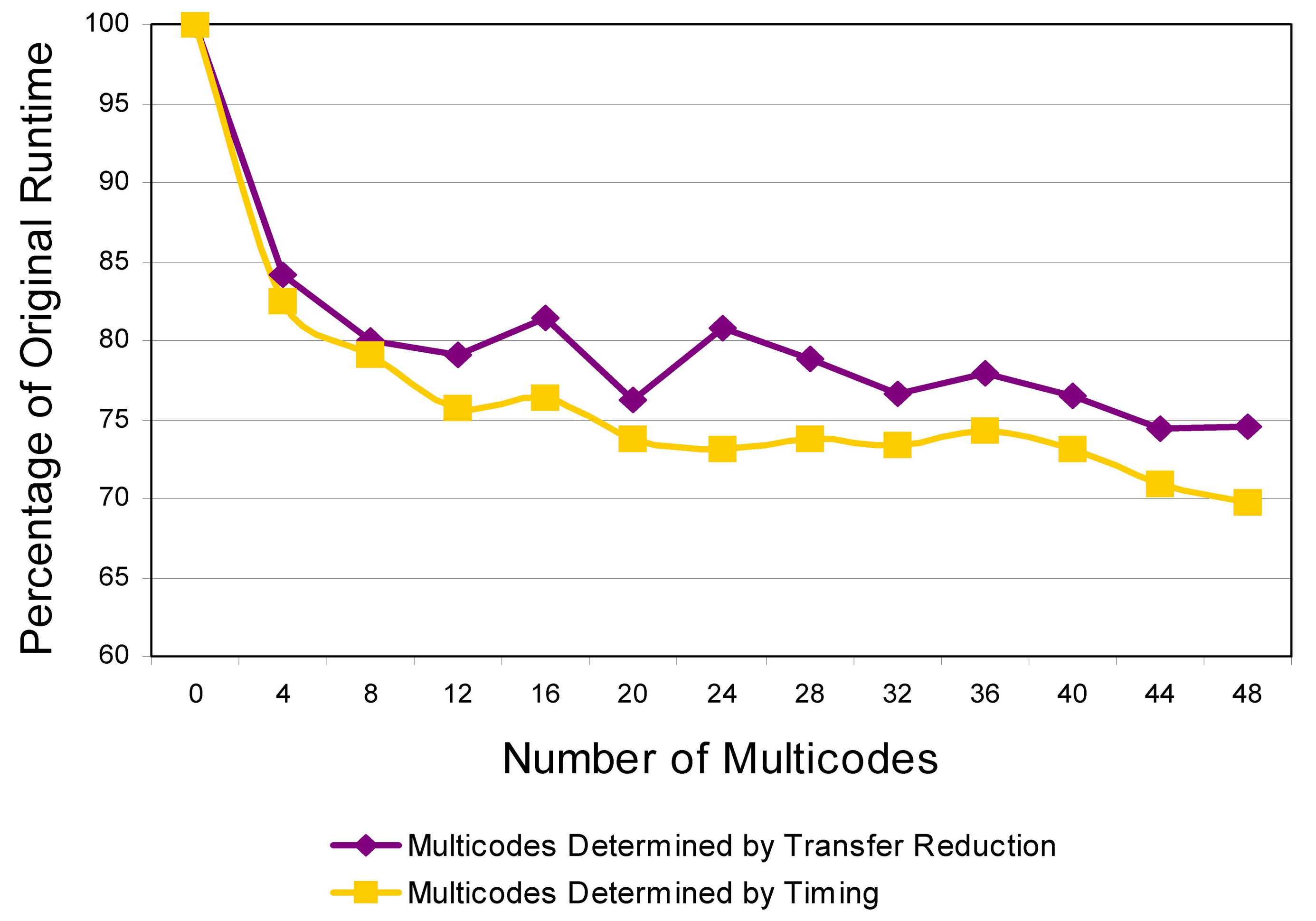
- Initially, the average length of the multicodes identified through timing is shorter than than those identified using transfer reduction alone.
- This trend reverses when more than 30 multicodes are identified for _222_mpegaudio

#	Len	Transfer Reduction Multicode	#	Len	Timing Multicode
1	50	iconst_0 faload fmul fadd fstore fload aload_0 getfield iload aaload iload bipush iadd faload aload iconst_1 faload fmul fadd fstore iload iconst_1 iadd bipush iand istore fload_3 aload_0 getfield iload aaload iload faload aload iconst_2 faload fmul fadd fstore_3 fload aload_0 getfield iload aaload iload faload aload iconst_3 faload fmul	1	50	iconst_0 faload fmul fadd fstore_3 fload aload_0 getfield iload aaload iload bipush iadd faload aload iconst_1 faload fmul fadd fstore fload aload_0 getfield iload aaload iload bipush iadd faload aload iconst_0 faload fmul fadd fstore fload aload_0 getfield iload aaload iload bipush iadd faload aload iconst_1 faload fmul fadd fstore
2	45	getstatic iload iinc aaload astore fload_3 aload_0 getfield iload aaload iload bipush iadd faload aload iconst_0 faload fmul fadd fstore_3 fload aload_0 getfield iload aaload iload bipush iadd faload aload iconst_1 faload fmul fadd fstore fload aload_0 getfield iload aaload iload bipush iadd faload aload	2	50	iconst_1 iadd bipush iand istore fload_3 aload_0 getfield iload aaload iload faload aload iconst_2 faload fmul fadd fstore_3 fload aload_0 getfield iload aaload iload faload aload iconst_3 faload fmul fadd fstore fload aload_0 getfield iload aaload iload faload aload iconst_2 faload fmul fadd fstore fload aload_0 getfield iload aaload iload
3	38	fadd fstore fload aload_0 getfield iload aaload iload faload aload iconst_2 faload fmul fadd fstore fload aload_0 getfield iload aaload iload faload aload iconst_3 faload fmul fadd fstore iload iconst_1 iadd bipush iand istore iinc iload bipush if_icmplt	3	10	faload fmul fadd fstore iload iconst_1 iadd bipush iand istore
4	27	aload_0 getfield aload_0 dup getfield dup_x1 iconst_1 iadd putfield saload istore aload_1 iload iinc iload sipush iand i2b bastore aload_1 iload iinc iload bipush iushr i2b bastore	4	27	aload_0 getfield aload_0 dup getfield dup_x1 iconst_1 iadd putfield saload istore aload_1 iload iinc iload sipush iand i2b bastore aload_1 iload iinc iload bipush iushr i2b bastore
5	2	aload_0 getfield	5	15	getstatic iload iinc aaload astore fload_3 aload_0 getfield iload aaload iload bipush iadd faload aload
6	37	aload_2 bipush dup2 faload aload_2 bipush faload fsub fastore aload_2 bipush dup2 faload aload_2 bipush faload fsub fastore aload_2 bipush dup2 faload aload_2 bipush faload fsub fastore aload_2 bipush dup2 faload aload_2 bipush faload fsub fastore aload_2	6	2	aload_0 getfield
7	50	bipush faload aload_1 bipush faload fsub getstatic bipush faload fmul putfield aload_0 aload_1 bipush faload aload_1 bipush faload fsub getstatic bipush faload fmul putfield aload_0 aload_1 bipush faload aload_1 bipush faload fsub getstatic bipush faload fmul putfield aload_0 aload_1 bipush faload aload_1 bipush faload fsub getstatic bipush faload fmul putfield	7	46	aload_2 bipush dup2 faload aload_2 bipush faload fsub fastore aload_2 bipush dup2 faload aload_2 bipush faload fsub fastore aload_2 bipush dup2 faload aload_2 bipush faload fsub fastore aload_2 bipush dup2 faload aload_2 bipush faload fsub fastore aload_2 bipush dup2 faload aload_2 bipush faload fsub fastore aload_2
8	27	aload_1 iload faload aload_2 iload faload fadd ldc fmul fstore aload_2 iload aload_1 iload faload aload_2 iload faload fsub ldc fmul fastore aload_1 iload fload fastore iinc	8	50	faload aload_1 bipush faload fsub getstatic bipush faload fmul putfield aload_0 aload_1 bipush faload aload_1 bipush faload fsub getstatic bipush faload fmul putfield aload_0 aload_1 bipush faload aload_1 bipush faload fsub getstatic bipush faload fmul putfield aload_0 aload_1 bipush faload aload_1 bipush faload fsub getstatic bipush faload fmul putfield aload_0

3 Performance Results

The timing based strategy consistently shows better performance results than using transfer reductions alone

- In the best case, transfer reductions achieve a speedup of approximately 25% while timing can be used to achieve a speedup of 30%



4 Conclusion

A framework is presented that utilizes timing during the multicode identification process. This has resulted in greater performance gains than previous multicode identification techniques. The gains are achieved because a timing based strategy considers all impacts of the multicode including its impact on caching, code locality, optimization potential of the multicode sequence and the cumulative impact of these factors over several multicode substitutions.

Performance gains of as much as 30% were achieved for _222_mpegaudio compared to its original runtime. In comparison, the previous transfer reduction strategy was only able to achieved gains of approximately 25%.

Acknowledgments

The authors would like to thank the Natural Sciences and Engineering Research Council of Canada for their support.

References

[1] B. Stephenson and W. Holst. Advancements in multicode optimization. In *OOPSLA '04: Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, pages 186–187, New York, NY, USA, 2004. ACM Press.